



Triggers (Gatilhos)

Os exemplos utilizados ao longo do texto, salvo menção em contrário, foram adaptados de GROFF, J. R., WEINBERG, P. N. e OPPEL, A. J. **SQL: The Complete Reference**, relacionado nas referências bibliográficas. Os *scripts* para criação e população das tabelas encontram-se no material de apoio.

O QUE SÃO *Triggers* (Gatilhos)

Triggers são programas armazenados que são disparados (daí o nome) quando determinados eventos ocorrem. Cinco tipos de eventos podem disparar um *trigger*:

- Execução de comandos DML (INSERT, UPDATE e DELETE) em tabelas. São chamados de gatilhos DML;
- Execução de comandos DDL (CREATE, ALTER, DROP). São chamados de gatilhos DML;
- Certos estados do SGBDR: inicialização (*start up*), desligamento (*shut down*), estabelecimento de conexão (*log in*), encerramento de conexão (*log out*) e ocorrência de erros;
- Execução de comandos DML (INSERT, UPDATE e DELETE) em *views*;
- Suspensão de execução de comandos, devido a esgotamento dos recursos autorizados para o usuário (espaço em disco, por exemplo).

No contexto de aplicações, *triggers* são muito úteis para garantir que regras de negócio estejam sendo obedecidas pelas aplicações.

TRIGGERS DML

Triggers DML são, de longe, os mais usados por desenvolvedores. Eles estão associados a tabelas e são disparados sempre que um comando DML é executado na respectiva tabela. *Triggers* podem ser disparados antes ou depois da execução do comando. Também podem ser disparados uma vez para cada comando (*triggers* no nível de comando ou *statement-level triggers*) ou a cada registro inserido/alterado/excluído (*triggers* no nível de registro ou *row-level triggers*). A definição de um *trigger* tem a seguinte forma geral:

```
CREATE [OR REPLACE] TRIGGER nome_trigger
{BEFORE | AFTER} {INSERT | DELETE | UPDATE | UPDATE OF lista_colunas} ON tabela
[FOR EACH ROW]
[FOLLOWS | PRECEDES nome_outro_trigger]
[ENABLE | DISABLE]
[WHEN (condição)]
[DECLARE
   declarações;]
BEGIN
   bloco_de_comandos;
[EXCEPTION
   bloco_de_tratamento_de_exceções;]
END [nome_trigger];
```

- O comando CREATE TRIGGER cria um novo *trigger* com nome *nome_trigger*. A cláusula opcional OR REPLACE substitui o *trigger*, caso ele exista;
- As cláusulas BEFORE e AFTER definem se o *trigger* é disparado antes ou depois o comando DML;

- As cláusulas INSERT, DELETE e UPDATE defir que comando DML causará o disparo. UPDATE ON deve ser utilizado no lugar de UPDATE caso se deseje que o disparo ocorra quando determinadas colunas forem alteradas (separadas por vírgulas). É possível definir mais de uma cláusula utilizando o operador OR;

- A cláusula FOR EACH ROW define um *trigger* no nível de registro, disparando-o para cada linha alterada. Deve-se omitir esta cláusula para *triggers* no nível de comando.

- Vários *triggers* podem ser definidos para um mesmo comando DML. FOLLOWS e PRECEDES definem a ordem de disparo;

- A cláusulas ENABLE e DISABLE determinam se o *trigger* estará inicialmente habilitado (ativo) ou desabilitado. Apenas *triggers* habilitados são disparados. Se não informado, o estado inicial é habilitado (ENABLE);

- WHEN define em que condições o *trigger* deve ser disparado, evitando que o bloco de comandos seja executado desnecessariamente. A condição deve vir entre parênteses;

- As cláusulas DECLARE, BEGIN e EXCEPTION têm a mesma função que em *procedures* e funções. Opcionalmente, pode-se incluir o nome do *trigger* após o END.

O estado de triggers DML pode ser alterado utilizando-se o comando ALTER TRIGGER:



```
ALTER TRIGGER nome_trigger {ENABLE | DISABLE}
```

O comando DROP TRIGGER *nome_trigger* exclui um *trigger* existente.

OS PSEUDOREGISTROS OLD E NEW

Triggers DML são disparados quando são feitas alterações na tabela a que estão vinculados. PL/SQL oferece duas estruturas, OLD e NEW, que são populadas imediatamente antes do disparo do *trigger*. OLD possui os valores (colunas) do registro afetado antes da execução do comando DML, enquanto NEW possui os valores que o registro afetado terá após a execução do comando DML.

Dependendo do comando DML que disparou o *trigger*, as seguintes situações podem ocorrer:

- INSERT: apenas a estrutura NEW possui valores válidos (não havia valores anteriores à inserção de um novo registro);
- DELETE: apenas a estrutura OLD possui valores válidos (não há novos valores após a exclusão de um registro);
- UPDATE: ambas as estruturas OLD e NEW possuem valores válidos.

Alguns cuidados com o uso destas estruturas:

- Deve-se colocar ':' antes de referências a OLD e NEW no bloco de comandos apenas [1] ;

- OLD e NEW só são criados em *triggers* no nível de linha. Referências a eles em *triggers* no nível de comando causarão erro.

- Não é possível modificar valores na estrutura OLD, mas isto é possível na estrutura NEW;

De forma geral, não é permitido que *triggers* no nível de linha consultem ou alterem a tabela a qual estão vinculados. Esta situação é apontada com o erro de compilação *ORA-04091 table xxx is mutating*. Para que sejam possíveis consultas, deve-se utilizar a diretiva PRAGMA AUTONOMOUS_TRANSACTION. *Triggers* no nível de comando são livres para consultar e alterar as respectivas tabelas associadas.

No exemplo a seguir o *trigger VERIFICA_META* é criado. Ele é disparado antes de alterações na coluna *meta* da tabela *REPRESENTANTES*. A cláusula WHEN habilita a execução apenas se a coluna tiver seu valor alterado. Este *trigger* limita alterações na meta de um represente a ±20% da média das metas dos representantes vinculados à mesma filial. Caso a meta seja 0 (o representante é novo e ainda não tem meta), permite a alteração para qualquer valor.

[1] OLD e NEW são estruturas do tipo RECORD, declaradas como *bind variables* (variáveis de ligação, em tradução literal). Para acessar *bind variables* dentro do bloco de comandos, deve-se utilizar o ':' como prefixo, porém isto já não é necessário (na verdade, não é permitido) quando a referência ocorre na cláusula WHEN.



```
CREATE OR REPLACE TRIGGER verifica_meta
AFTER UPDATE OF meta ON representantes
FOR EACH ROW
/* Executa o bloco de comandos apenas se meta for alterada       */
WHEN ((:OLD.meta != :NEW.meta) OR (:NEW.meta = 0))
DECLARE
    media NUMBER;
    /* Permite que a tabela associada ao trigger seja consultada */
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    /* Exclui do cálculo da média os registros com meta = 0      */
    SELECT AVG(R.meta) INTO media
    FROM representantes R
    WHERE R.filial_id = :NEW.filial_id AND R.meta <> 0;
    /* Se a consulta não retornar registros para o calculo da
       média, media := NULL                                      */
    IF media IS NOT NULL AND ABS((:OLD.meta - :NEW.meta) / :OLD.meta) > 0.2 THEN
        RAISE_APPLICATION_ERROR(-20000, 'Variação da meta excede 20%');
    END IF;
END;

UPDATE representantes set meta = 500000 where rep_id = 102;
```

O resultado apresentado após a execução dos comandos UPDATE é:

```
Error starting at line : 1 in command -
UPDATE representantes set meta = 500000 where rep_id = 102
Error report -
ORA-20000: Variação da meta excede 20%
ORA-06512: at "VERIFICA_META", line 13
ORA-04088: error during execution of trigger 'C##SUPER.VERIFICA_META'

1 row updated.
```

IDENTIFICANDO O EVENTO QUE DISPAROU O *TRIGGER*

A linguagem PL/SQL possui um conjunto de funções que permitem identificar o comando DML que disparou o *trigger*. São as funções INSERTING, DELETING e UPDATING. Elas permitem que as ações certas sejam tomadas em função do tipo de alteração que está sendo efetuada na tabela. O *trigger VERIFICA_GERENTE*, mostrado a seguir, verifica se os campos *id_rep* e *gerente* são iguais

em inclusões e alterações de registros na tabela *REPRESENTANTES*. Se for um comando INSERT, a operação é rejeitada com uma mensagem de erro. Para alterações, o campo *gerente* recebe o valor NULL.

```
CREATE OR REPLACE TRIGGER verifica_gerente
BEFORE INSERT OR UPDATE ON representantes
FOR EACH ROW
/* Executa o bloco de comandos apenas se gerente = rep_id */
WHEN (NEW.rep_id = NEW.gerente)
BEGIN
    IF UPDATING THEN
        RAISE_APPLICATION_ERROR(-20000, 'Os campos rep_id e gerente não podem ser iguais');
    ELSE
        :NEW.gerente := NULL;
    END IF;
END;

INSERT INTO representantes VALUES (130, 'John Doe', 33, 21, 'Rep. Comercial', '20-03-03', 130, 100000,
0);
UPDATE representantes SET gerente = 130 WHERE rep_id = 130;
```

O registro incluído tem os seguintes valores:

		NOME	IDADE	FILIAL_ID	DATA_I...	GER...	META	VENDAS	CARGO
1	130	John Doe	33	21	20/03/03	(null)	100000	0	Rep. Comercial

Após a execução do comando UPDATE, o resultado é:

```
Error starting at line : 15 in command -
UPDATE representantes SET gerente = 130 WHERE rep_id = 130
Error report -
ORA-20000: Os campos rep_id e gerente não podem ser iguais
ORA-06512: at "VERIFICA_GERENTE", line 3
ORA-04088: error during execution of trigger 'VERIFICA_GERENTE'
```

DEFININDO A ORDEM DE DISPARO DE *TRIGGERS*

É possível definir mais de um *trigger* para uma mesma condição de disparo. Para garantir a ordem em que serão disparados, utilizam-se as cláusulas FOLLOWS e PRECEDES. Estas cláusulas fazem com

que o *trigger* em questão seja disparado após/antes o *trigger* cujo identificador aparece logo em seguida a elas.

No exemplo a seguir, são definidos dois *triggers* para a tabela *PEDIDOS.* O primeiro, *VERIFICA_QTD*, não permite que sejam colocados pedidos com quantidades superiores aos estoques dos respectivos produtos. O segundo, *VERIFICA_LIMITE_CRED*, impede que sejam colocados pedidos cujo valor total exceda o limite de crédito do respectivo cliente. Você deseja que *VERIFICA_QTD* seja disparado antes de *VERIFICA_LIMITE_CRED.*

```
CREATE OR REPLACE TRIGGER verifica_qtd
BEFORE INSERT ON pedidos
FOR EACH ROW
DECLARE
    qtd NUMBER;
BEGIN
    SELECT P.qtd_disponivel INTO qtd FROM produtos P
    WHERE P.fornec_id = :NEW.fornec_id AND P.produto_id = :NEW.produto_id;
    IF qtd < :NEW.qtd THEN
        RAISE_APPLICATION_ERROR(-20000,'Quantidade insuficiente do produto ' ||
          :NEW.produto_id || ' fornecido por ' || :NEW.fornec_id);
    END IF;
END;

CREATE OR REPLACE TRIGGER verifica_limite_cred
BEFORE INSERT ON pedidos
FOR EACH ROW
FOLLOWS verifica_qtd
DECLARE
    limite_credito NUMBER;
    empresa varchar2(20);
BEGIN
    SELECT C.limite_credito, C.empresa INTO limite_credito, empresa FROM clientes C
    WHERE C.cliente_id = :NEW.cliente_id;
    IF limite_credito < :NEW.qtd * :NEW.valor THEN
        RAISE_APPLICATION_ERROR(-20000,'Limite de crédito insuficiente para o cliente ' || empresa);
    END IF;
END;

INSERT INTO pedidos (cliente_id, data, fornec_id, pedido_id, produto_id, qtd, rep_id, valor)
VALUES (2114, '20/03/03', 'BIC', 114000, '41003', 10, 103, 100.00);

INSERT INTO pedidos (cliente_id, data, fornec_id, pedido_id, produto_id, qtd, rep_id, valor)
VALUES (2114, '20/03/03', 'ACI', 114000, '41003', 200, 103, 125.00);

INSERT INTO pedidos (cliente_id, data, fornec_id, pedido_id, produto_id, qtd, rep_id, valor)
VALUES (2118, '20/03/03', 'ACI', 114000, '41003', 200, 103, 125.00);
```

Após a execução dos três comandos de inserção, o resultado é:

```
ORA-20000: Quantidade insuficiente do produto 41003 fornecido por BIC
ORA-06512: at "VERIFICA_QTD", line 6
ORA-04088: error during execution of trigger 'VERIFICA_QTD'

ORA-20000: Limite de crédito insuficiente para o cliente Orion Corp.
ORA-06512: at "C##SUPER.VERIFICA_LIMITE_CRED", line 7
ORA-04088: error during execution of trigger 'C##SUPER.VERIFICA_LIMITE_CRED'

1 row inserted.
```



TRIGGERS DDL

Triggers DDL são muito similares a seus primos DML, porém são disparados a partir da execução de comandos DDL. Outra diferença importante diz respeito ao escopo do *trigger*. Enquanto *triggers* DML estão associados a uma tabela específica, *triggers* DDL são associados ao esquema em que foram criados ou todo o banco de dados. Sua forma geral é:[1]

[1] Qualquer comando DDL, como GRANT, REVOKE e AUDIT, pode ser usado com evento de disparo. Os mais comuns são CREATE, ALTER e DROP. Para definir a condição de disparo para qualquer comando DDL, utilize o identificador DDL.

```
CREATE [OR REPLACE] TRIGGER nome_trigger
{BEFORE | AFTER}
{commando DDL, commando_DDL, ...} ON {DATABASE | SCHEMA}
[WHEN (condição)]
[DECLARE
    declarações;]
BEGIN
    bloco_de_comandos;
[EXCEPTION
    bloco_de_tratamento_de_exceções;]
END [nome_trigger];
```

As demais cláusulas funcionam como nos t*riggers* DML. Exemplo:



```
CREATE OR REPLACE TRIGGER vigia
AFTER RENAME ON SCHEMA
BEGIN
    DBMS_OUTPUT.PUT_LINE('Algo trocou de nome');
END;

RENAME representantes TO tab_rep;
```

O resultado, após a execução do comando RENAME, é:

```
Algo trocou de nome
```

Como *triggers* DDL são vinculados a um esquema particular ou a todo o banco de dados, é importante saber, além do comando DDL causou o disparo (condições de disparo DDL ou com o operador OR), que objeto foi afetado pelo comando. Para tal, há um conjunto de funções úteis, chamadas de funções de atributos de eventos (*event atribute functions*). As mais comuns são mostradas Tabela 1.[1]

[1] O conjunto completo de funções pode ser encontrado em https://docs.oracle.com/cd/B10501_01/appdev.920/a96590/adg14evt.htm.

Função	Descrição
ORA_DATABASE_NAME	Nome do banco de dados onde o *trigger* foi disparado.
ORA_DICT_OBJ_NAME	Nome do objeto afetado pelo comando que disparou o *trigger*.
ORA_DICT_OBJ_OWNER	Dono (criador) do objeto afetado pelo comando DDL.
ORA_DICT_OBJ_TYPE	Tipo do objeto afetado pelo comando DDL.
ORA_LOGIN_USER	Usuário da conexão (sessão) onde o comando DDL foi executado.
ORA_SYSEVENT	Comando DDL que causou o disparo.
ORA_IS_ALTER_COLUMN	Retorna TRUE se a coluna especificada está sendo alterada.
ORA_IS_DROP_COLUMN	Retorna TRUE se a coluna especificada está sendo excluída.

Tabela 1

A seguir, são apresentados diversos exemplos uso destas funções. As funções ORA_DICT_OBJ_TYPE e ORA_DICT_OBJ_NAME retornam o tipo e o nome do objeto afetado, respectivamente.

```
CREATE OR REPLACE TRIGGER vigia2
AFTER RENAME ON SCHEMA
FOLLOWS vigia
BEGIN
    DBMS_OUTPUT.PUT_LINE('O comando RENAME ' || ORA_DICT_OBJ_TYPE || ' ' ||
        ORA_DICT_OBJ_NAME || ' foi executado');
END;

RENAME representantes TO tab_rep;
```

O resultado, após a execução do comando RENAME, é:

```
Algo trocou de nome
O comando RENAME TABLE REPRESENTANTES foi executado
```

As funções ORA_LOGIN_USER e ORA_DICT_OBJ_OWNER retornam o usuário da conexão (sessão) e o usuário que criou o objeto afetado, respectivamente.

```
CREATE OR REPLACE TRIGGER vigia3
AFTER RENAME ON SCHEMA
FOLLOWS vigia2
BEGIN
    DBMS_OUTPUT.PUT_LINE('O comando acima foi executado pelo usuário ' || ORA_LOGIN_USER ||
        ' em um objeto criado pelo usuário ' || ORA_DICT_OBJ_OWNER);
END;

RENAME representantes TO tab_rep;
```

O resultado, após a execução do comando RENAME, é:

```
Algo trocou de nome
O comando RENAME TABLE ENDERECOS foi executado
O comando acima foi executado pelo usuário C##SUPER em um objeto criado pelo usuário C#
```



Duas funções demandam maior atenção: ORA_IS_ALTER_COLUMN e ORA_IS_DROP_COLUMN. Elas permitem que seja feito o controle de exclusões e alterações em colunas de uma tabela. No exemplo a seguir, o *trigger VERIFICA_ALTER_DROP_COLUMN* impede que as colunas das tabelas utilizadas nos exemplos sejam alteradas ou excluídas.

```
CREATE OR REPLACE TRIGGER verifica_alter_drop_column
BEFORE ALTER OR DROP ON SCHEMA
   WHEN ((ORA_DICT_OBJ_TYPE = 'TABLE') AND (ORA_DICT_OBJ_NAME IN ('REPRESENTANTES', 'PRODUTOS',
   'CLIENTES', 'PEDIDOS', 'FILIAIS')))
DECLARE
    tabela VARCHAR2(20);
BEGIN
    tabela  := ORA_DICT_OBJ_NAME;
    FOR coluna IN (SELECT column_name FROM all_tab_columns WHERE table_name = tabela) LOOP
        IF ORA_IS_ALTER_COLUMN(coluna.column_name) THEN
             RAISE_APPLICATION_ERROR(-20001, 'Não é permitido alterar a coluna ' || coluna.column_name
                || ' da tabela ' || tabela);
        ELSIF ORA_IS_DROP_COLUMN(coluna.column_name) THEN
             RAISE_APPLICATION_ERROR(-20001, 'Não é permitido excluir a coluna ' || coluna.column_name
                || ' da tabela ' || tabela);
        END IF;
    END LOOP;
END;
```

O comando FOR percorre todas as colunas da tabela afetada verificando se serão alteradas ou excluídas.

Atividade Complementar

Em geral, não é permitido que um *trigger* DML consulte dados da tabela a qual está vinculado por causa de um fenômeno chamado *dirty reads* (leituras sujas). No entanto, há uma forma de fazê-lo. Pesquise o que é e como podem ser evitadas leituras sujas no SGBDR Oracle.



Referência Bibliográfica

- **FEUERSTEIN, S. Oracle PL/SQL Programming. 6ª Ed., O'Reilly, 2014.**

- PUGA, S., FRANÇA, E. e GOYA, M. **Banco de Dados: Implementação em SQL, PL SQL e Oracle 11g**. São Paulo: Pearson, 2014.

- Gonçalves, E. **PL/SQL: Domine a linguagem do banco de dados Oracle**. Versão Digital. Casa do Código, 2015.

- GROFF, J. R., WEINBERG, P. N. e OPPEL, A. J. **SQL: The Complete Reference.** 3ª Ed., Nova York: McGraw-Hill, 2009.

- ELMASRI, R. e NAVATHE, S. B. **Sistemas de Banco de Dados**. 7ª Ed., São Paulo: Pearson, 2011.

Ir para exercício